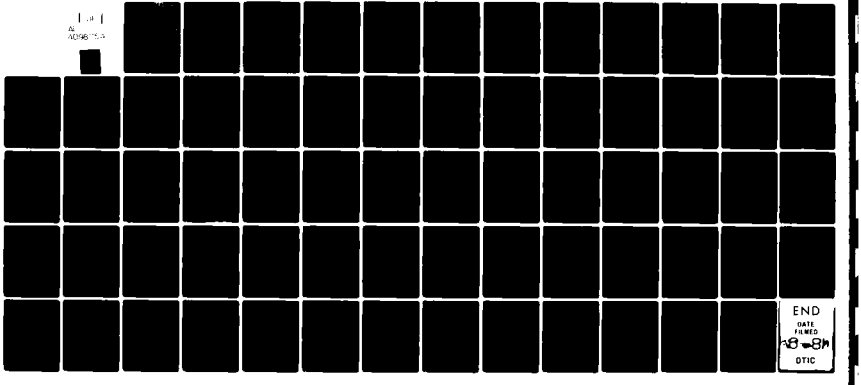
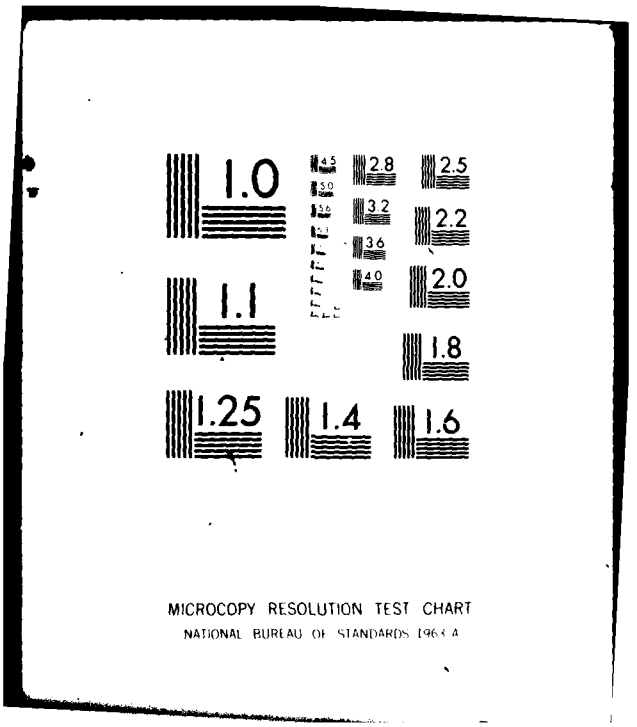
AD-A098 754 STUTTGART UNIV (GERMANY F R) INST FUER INFORMATIK F/G 9/2
COMMUNICATION INTERFACE FOR A SYSTEM OF DISTRIBUTED PROCESSES.(U)
MAR 81 E J NEUHOLD, K BOEHME DAERO-79-G-0008

UNCLASSIFIED NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

COMMUNICATION INTERFACE FOR A SYSTEM OF DISTRIBUTED PROCESSES

Final Report

E.J. Neuhold
and
K. Böhme
Institut für Informatik
University of Stuttgart
Azenbergstr. 12
D-7000 Stuttgart 1
Fed. Rep. of Germany

31.3.1981

Grant DAERO-79-G-0008 (R&D 2628)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
	AD-A098 754	

4. TITLE (and Subtitle)
COMMUNICATION INTERFACE FOR A SYSTEM OF DISTRIBUTED PROCESSES.

5. TYPE OF REPORT & PERIOD COVERED
FINAL (31.3.80-31.3.81)

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
E.J. NEUHOLD and K. BOEHME

8. CONTRACT OR GRANT NUMBER(s)
DAERO-79-G-0008

9. PERFORMING ORGANIZATION NAME AND ADDRESS
INSTITUT FUER INFORMATIK
UNIVERSITY OF STUTTGART, AZENBERGSTRASSE 12
D-7000 STUTTGART 1, GERMANY

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
1T161102BH57-03

11. CONTROLLING OFFICE NAME AND ADDRESS
EUROPEAN RESEARCH OFFICE
U.S. ARMY RESEARCH, DEVELOPMENT & STANDARDIZATION GP (EUR), Box 65, NY 09510

12. REPORT DATE
31.3.1981

13. NUMBER OF PAGES
66

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

15. SECURITY CLASS. (of this report)
UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release, distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

N/A

18. SUPPLEMENTARY NOTES

The view, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Inter process communications
Distributed processes
Protocol
Petri-nets
X-25
Packet switching
heterogeneous computer networks

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Interprocess communication (IPC) has been investigated. The model is a process consisting of 7 layers (as proposed by ISO/TC97/SC16), which has been implemented. The layers 1 - 3 build the interface to a X.25 packet switching network. Their function has been specified by CCITT. As these parts are incompletely and incorrectly described, solutions have to be found to obtain a complete interface. New algorithms have been developed in the Transport Layer: end-to-end error control, in the Session Layer: connection synchronization, ...

Final Report

The results of this research project have been described in the quarterly progress reports and mainly in the paper "The Layers 4 to 7 of an Interprocess Communication System - Implementation Aspects", which was supplied with the 5th quarterly progress report. Therefore the final report consists of 3 parts:

- a brief summary
- 2 papers "An Interprocess Communication Service Applied in a Distributed Data Base System" (submitted to ECI, München 10/81) and "The Transport Service of an Interprocess Communication System (submitted to 7th Data Communication Symposium, Mexico City, 10/81)
- a description of the user's interface

What has been achieved?

Interprocess communication (IPC) has been investigated. The result is a system consisting of 7 layers (as proposed by ISO/TC97/SC16), which has been implemented. The layers 1 - 3 build the interface to a X.25 packet switching network. Their function has been specified by CCITT. As some parts are incompletely and incorrectly described, solutions have to be found to obtain a complete interface. New algorithms have been developed in the Transport Layer: end-to-end error control, in the Session Layer: connection establishment and release and in the Presentation Layer: intelligent data converter.

In the area of protocol verification Petri-Nets, Hoare's CSP and Lauer's COSY have been studied.

The 2-year research project was the basis for two new research projects, one supported by ERO: "Application Layer Protocols for DDBS" and the other supported by DFG: "Process Interactions in a Distributed Data Base System". It was further the basis for collaboration with the German PIX research group and with WG1 of ISO/TC97/SC16.

Implementations

All implementations on the PDP11 are made under the operating system RSX11M V3.1 in PASCAL and Assembler. Programs for the LSI11 have been developed on the PDP11 (also in PASCAL and Assembler). The LSI11 runs without operating system.

- Connection TR440-PDP11/40:
  3 TR440 terminals can be simulated on the PDP; the software is interrupt driven; filetransfer in both directions is possible

- Connections PDP11/44-PDP11/40, PDP11-LSI11, LSI11-LSI11:
  own-written full-duplex driver based on a asynchronous line interface

- Connection PDP11-LSI11:
  realization of a hardware connection with direct-memory-access interfaces

- LSI11:
  - bootstrap loader to load any program developed on a PDP11
  - mini operating system to schedule processes in dependence of timer or of other processes
  - IPC:
    layers 6 - 3 are implemented on the PDP11,
    the X.25 interface is almost implemented on the LSI11

Reports and Papers

1) Böhme, K., Peter, G.:
Process Communication Structures for Distributed Systems.
Proc. GI-Workshop "Kommunikation in verteilten Datenbanksystemen", Berlin, 12/79

2) Böhme, K.:
The Connection TR440-PDP11/40.
Techn. Report No. 1/80, Inst. für Informatik, Univ. of Stuttgart, 1980

3) Böhme, K.:
The Layers 4 to 7 of an Interprocess Communication System - Implementation Aspects -.
Techn. Report No. 3/80, Inst. für Informatik, Univ. of Stuttgart, 1980

4) Böhme, K.:
An Interprocess Communication Service Applied in a Distributed Data Base System.
Submitted to ECI, München, Oct. 1981

5) Böhme, K.:
The Transport Service of an Interprocess Communication System.
Submitted to 7th Data Communication Symposium, Mexico City, Oct. 1981

The following Diplomarbeiten (equivalent to master thesis) and Studienarbeiten ("pre"-master thesis) have been worked out during the period of the grand (all in German):

1) Rauther, I.:	Integration of a Data Converter
2) Kißling, M.:	Service Routines for the Communication Processor LSI11
3) Hermann, H.:	Data Transfer with DMA between PDP11 and LSI11
4) Merdes, H.:	Use of a LSI11 as a Communication Processor
5) Gelpke, D.:	Nondeterministic Message Exchange
6) König, P.:	Implementation of X.25 on an LSI11 (not yet finished)

AN INTERPROCESS COMMUNICATION SERVICE APPLIED IN A DISTRIBUTED DATA BASE SYSTEM

Klaus Böhme
Institut fuer Informatik
University of Stuttgart
Azenbergstrasse 12
D-7000 Stuttgart 1
Fed. Rep. of Germany

Abstract:

An IPC system for a heterogeneous computer network is described. It is pointed out how the system meets the requirements of process communication in the Distributed Data Base System POREL. The IPC system is related to ISO's model of Open Systems Interconnection.

Characteristics of the system are: Reliable data transport based on connections, conversion of data items and data structures, support for realizing complex systems of communicating processes since no asynchronous behaviour of application processes is required and since processes can wait for certain events.

This work has been supported by ERO Grant No. DAERO-79-G-0008.

1. Introduction

CS is an interprocess communication (IPC) system which is used in the distributed data base management system (DDBMS) POREL. (POREL is an experimental DDBMS implemented at the University of Stuttgart). The services and the behaviour of CS are designed to meet not only the requirements of a distributed data base system, but also for other applications which are based on communicating processes. Design criteria for POREL /see e.g. EN79, PO78/, however, had to be reflected in the design of CS and shall briefly be summarized.

Firstly, the underlaying computer network is heterogeneous. As a consequence, software had to be implemented on computers of different structure and size. To minimize the implementation effort of adapting all modules to special machines, the software has to be portable. An acceptably portable system was achieved by using a high level programming language (PASCAL) for implementation and leaving the operating systems untouched. The still necessary machine dependent code, (e.g. for special file I/O, for local event handling, for accessing communication devices, etc.) has been realized by small assembler routines with well defined interfaces. Only these routines have to be rewritten, if a new computer type shall be added to the network.

Secondly, for easy application CS has one interface only for local communication (within one computer system) and for remote communication (using the network).

Thirdly, there are performance and reliability constraints. As long as communication by means of a network is considerably slower than within a computer, protocols have to be designed needing a minimal number of messages. Therefore an error control mechanism was designed mainly based on timers (which do not bother the network) and which needs no

additional messages in the error-free case. If, however, an error has occurred, there exists a reliable resynchronization mechanism. For details see /Boe80/.

Good performance is also obtained, since users of the CS may serve many connections in parallel and are not unnecessarily blocked by the CS. They may get back control after the local part of CS has accepted or rejected a request. In most cases, no remote actions are required for that decision.

The services of CS support - to a certain degree - the reliability of communicating processes. Application programmers can use an "wait for an event"-service to synchronize processes and to get restart points in case of errors and therefore to realize transparent communication structures.

Although CS was designed before ISO introduced its reference model for Open Systems Interconnection /ISO79/, it has a similar layered structure. Layering was chosen

- because of its clear separation of different functions,
- because a certain layer can be described by abstracting from the underlying layer
- because it is easy to adapt a layer to new circumstances or to add new functions to a layer.

The advantages of layering are well-known and are similar to those of modularity. (A layered system is modular, but the reverse is not true).

Relation to Previous Work

The design of CS was considerably influenced by the collaboration with PIX. PIX specified computer and application independent higher level protocols which are suitable for standardization /BV78, Vo79/.

The notion of service-primitives was introduced by Bochmann and Vogt in 1978 /BV78/. It is essential for the

understanding of the distributed execution of requests in a layered system.

Watson and Fletcher describe in /WF79/ their network operating system which is datagram- and not connection-oriented. Their timer based protocol and the comparison with other protocol mechanisms gave valuable impulses for the design of the reliable end-to-end protocol of CS.

Structure of this paper

Chapter 2 describes the architecture of CS. In chapter 3 the requirements of the users of CS (the Application Layer) are stated and implementation decisions are discussed, which meet the requirements. Chapter 4 introduces the Presentation Layer which is responsible for data conversion. Chapter 5 describes the Session Layer. This is the lowest layer which is concerned in application requirements. The layers below are descibed in another paper /Boe80/.

2. The Architecture

Although ISO's Model for Open Systems Interconnection still is in an evolutionary state, it seems useful that communication experts accept its suggested terminology and layering concept and will thus describe their own systems in a uniform way.

The layered homogeneous architecture of CS is illustrated in fig. 2.1. There we have 3 application entities (P_1, P_2, P_3) distributed over 2 sites having local and remote connections.

To better understand the layering concept, some definitions given by ISO shall be repeated here and applied to the architecture of CS. All functions of CS can be arranged (according to ISO) so that we obtain a hierarchy of six layers. The user of CS forms the seventh layer, the Application Layer. The layer below, layer 6, is the Presentation Layer which converts user data, if necessary. The transport of data is based on associations between application entities. The purpose of the Session Layer (layer 5) is to organize and synchronize the dialogue of these application entities. The Transport Layer (layer 4) transfers data in a reliable and cost effective way by using the available communication resources. In CS the Network Interface (layers 3, 2 and 1) is defined by X.25.

Each layer consists of several entities which realize its associated functions possibly by cooperating with other entities of the same layer (if the function requires distributed actions) and by using functions provided by the underlying layer.

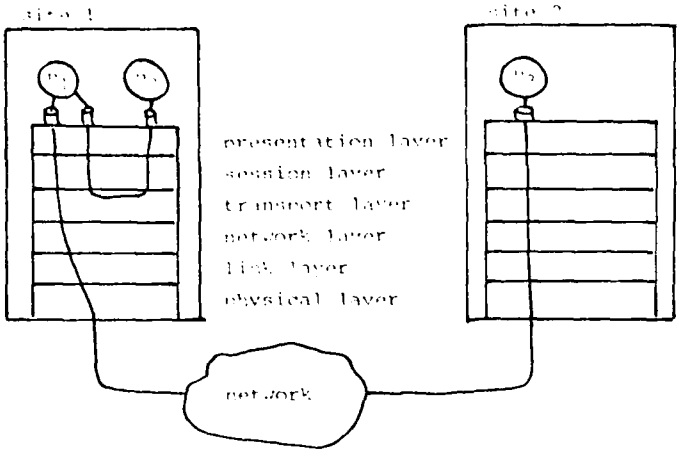
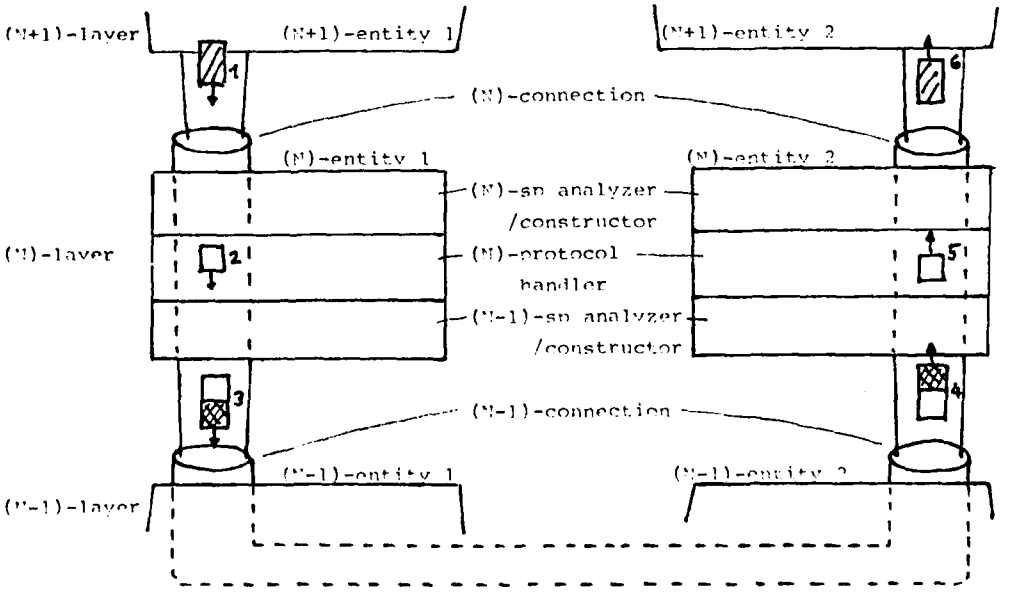


Fig. 2.1 layered architecture of OS



sp ... service primitive
... (N)-service-primitive-request
... (N)-protocol-data-unit
... (N-1)-service-primitive-request

the numbers indicate points of time

Fig. 2.2: Mapping and flow of service-primitives and protocol-data-units

Each (N)-layer provides entities in the (N+1)-layer with (N)-services which can be accessed and are described by (N)-service-primitives. In a distributed system it may be necessary that entities of the same layer ((N+1)-layer, say), have to cooperate to execute a certain function. This is done by using an (N+1)-protocol and by using an (N)-service, which establishes and maintains (N)-connections between cooperating (N+1)-entities for the exchange of protocol data-units. If a function can locally be realized (involving 1 entity only), no protocol and no connection are necessary.

Note: Communication between adjacent layers is done by means of service-primitives, intralayer communication by protocols. Only the latter needs to be standardized in an open system, because different computer systems are involved. The former may be implemented at one's convenience.

Each entity consists of 3 functional units (cf. fig. 2.2): a service providing part at the upper end of the layer, a protocol part in the middle, and at the lower end a part which uses the underlying services.

A (N)-service-primitive request (from the (N+1)-layer) may cause that some (N)-protocol-data-units are generated. Every protocol-data-unit is forwarded within an (N-1)-service-primitive (see fig. 2.2). If no protocol-data-units have to be generated, the (N)-service-primitive request may be mapped into one (N-1)-service-primitive or it is locally treated by the N-entity itself.

3. Application Layer

The Application Layer (layer 7) is the user of CS. No assumptions are made by CS about layer 7-protocol characteristics. I.e. there may be flow-control mechanisms, error recovery mechanisms, application entities may work deadlock free or not. CS is independent of the correct working of application entities. Mechanisms to define reliable systems of communicating processes shall not be treated here. This is the field of semantics of concurrent programs and of protocol verification methods.

Nevertheless CS should have some properties to facilitate the design of reliable systems of communicating processes:

- errorfree, insequence and guaranteed delivery of application data (messages, files),
- notification, if there are errors which are not recoverable by CS (breakdown of sites or of all communication lines to one site),
- facility that application entities may synchronize themselves by certain events (e.g. to get reliable restart points),
- buffering of requests until the peer application entity wants to receive them.

As we shall see in this chapter, CS has these properties. The last property shall further be explained.

If all application requests are buffered, no asynchronous behaviour is required by the application entities. In other words, they do not have to receive messages or other information "at any time", but CS delivers them when the application-entities are willing to accept them. This essentially simplifies the structure of application-entities, since interrupt handling is not needed. (From operating systems it is well-known that interrupt structures are difficult to analyze and to test because it is nearly impossible to reproduce a certain behaviour).

3.1 Services Required from CS

An important design decision is: do we need connections between application entities or is it sufficient to offer a datagram-service? (A datagram is a piece of user data together with its destination address). The main advantage of datagrams is that no connections have to be established prior to data exchange. The main disadvantage is, however, that consecutively sent datagrams are not related with each other whereas data sent within a connection is time related, their sequence is maintained.

Fletcher and Watson /WF79/ are advocates of a datagram-oriented service. They say that the communication structure of most applications is merely a request/response scheme with no need for additional communication and therefore all the overhead of connection establishment and release is not justified.

In PORTL we also have request/reply structures /BP79/, so a datagram-service seems to be appropriate. But requests as well as replies may consist of one or more control messages and one or more input (output) files which have to be kept in sequence. Therefore, from an application programmer's view, who has to deal with recovery and resets, it seems easier and leads to more transparent structures if communication activities are bracketed by CONNECT and DISCONNECT and if insequence delivery is guaranteed between these brackets.

Summarizing the properties which we have mentioned in the previous sections, we now can list the requirements of the application layer (L7):

- Connection establishment/release between L7-entities
- Addressing of L7-entities and L7-connections
- Errorfree transport of L7-data (messages, files)
- Flow control mechansims
- Several classes of data priorities

- Conversion of L7-data, if L7-entities have different representations for data structures
- No asynchronous events, i.e. all service-primitives have to be ↓SP (downwards service-primitive, from L7 to the underlying layers) (cf. ch. 5)
- Facility of waiting for certain events
- Information about messages and about the state of the connection.

These services are implemented by offering service-primitives

- for connection handling: CONNECT, DISCONNECT,
- for message exchange: TRANSMIT, AWAIT,
- for file exchange: TRANSMITF, AWAITF,
- for getting information: INFORM
- for declaring data structures: DECLF.

A complete list of the service primitives and their parameters is given in /Boe80/.

3.2 Addressing

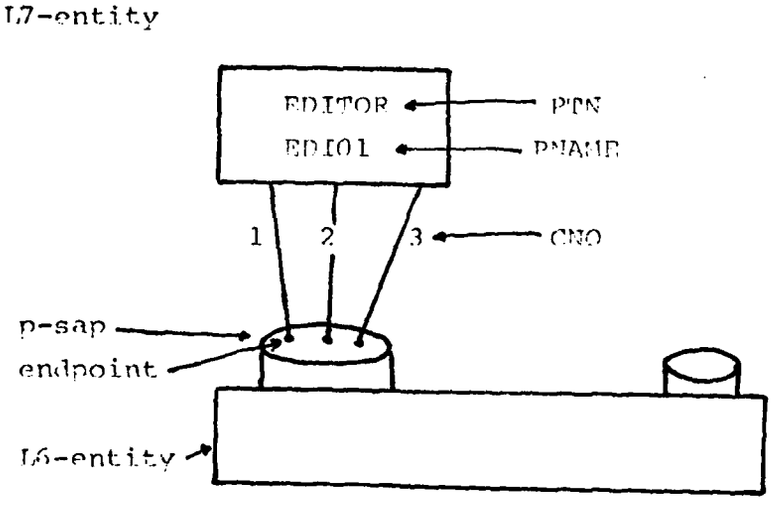
Naming and addressing is a topic mostly treated in an ad hoc manner. This especially is true for distributed systems. A good discussion of the problems in this field is given by Watson /Wa80/. ISO was engaged in this area, too. In its reference model /IS079/ it mainly clarifies the related terminology, which we will use to describe the naming policy within CS.

Application entities uniquely have to be addressed within the scope of the whole distributed system. This implementation (fig.3.1) uses the process-name (PNAME) which is provided by the operating system (every entity is a process) together with an identifier of the computer system (NODE) as a unique address of an application-entity. (More precisely PNAME is a mapping of the operating system's process-name). PNAME and NODE are machine-oriented names.

A name convenient for people is the process-type-name (PTN) which is associated with each application-entity. This name is chosen by the application programmer. If an application-entity becomes alive, it gets a PNAME and can be regarded as an instance of a certain process-type. For the CS PTN is a special sort of a generic process name. Before a new instance of a process-type is created (i.e. before an application process is started), only its PTN is known but not its PNAME, which it only will get from the operating system.

As application-entities can maintain several connections in parallel, these connections have to be distinguished. We use connection-numbers (CNO) to uniquely identify connections within the scope of the application-entity.

The triple NODE.PNAME.CNO forms a hierarchic name which uniquely identifies a connection in the distributed system. The



PTN ..... process-type-name
PNAME ..... process-name
CNO ..... connection number
p-sap ..... presentation-service-access-point

Fig. 3.1: Example of addressing L6-connections and an L7-entity

relation to ISO's naming and addressing scheme shall now be explained.

In ISO's model an (N)-entity is attached to an (N-1)--service-access-point and can be addressed by the (N-1)--address of this service-access-point. Refer to fig. 3.1 for an example where the application-entity EDI01 has 3 connections. We assume that the presentation-service-access-point-address is EDI01 as well. Several connections within one service-access-point are distinguished by (N-1)-connection--endpoint-identifiers which, however, are an agreement only between both related entities (of adjacent layers) and are not part of the address. In our picture these identifiers are the numbers 1, 2 and 3.

If an (N)-entity wants to establish an (N-1)-connection to another (N)-entity it has to know the remote (N-1)-address. As in CS an application-entity may address a certain connection of a remote entity at connection establishment, the triple NODE.PNAME.CNO is taken as a presentation-address identifying a presentation-service-access-point.

For the DDBMS POREL this pure process (or entity) oriented naming method is not very well suited. Communication does not take place between any entities but between those working for a certain transaction. A transaction oriented data base system needs services like

- establish a connection to that entity which is of type PTN and works for transaction TNO
- send message m to all entities working for transaction TNO.

In POREL the generic name PTN therefore was extended and may identify a process type as well as a transaction.

4. Presentation layer

In an interprocess communication system the presentation layer (L6) has to manage conversion problems which arise from different computer systems or different programming languages. Below L6 data is independent of such differences. The application and the implementation language mainly determine the occuring data structures which have to be converted. Examples of structures (of increasing complexity) are

- integer, real, decimal, character, boolean, subrange
- array, record, string
- sequential file, random file, list
- output line of a printer, page of a terminal.

To convert data, L6 has to know the structure. It can be explicitly or implicitly defined.

By explicit definition, L7-entities describe the structure of a data-unit before or at transmission (e.g.: '3 integer, 6 character, ...'). Structure descriptions and values of data-units are separately maintained.

By implicit definition, values and structure descriptions are mixed. A message or file consists of data-units which have the following form:

$$\langle \text{data-unit} \rangle ::= \langle \text{type} \rangle \langle \text{lng} \rangle \langle \text{value}_1, \ \ldots \ , \text{value}_{\text{lng}} \rangle .$$

Which one of the 2 methods is used depends on the L7-entities. If they need transmitted data together with their structure description, implicit definition is advantageous. If only a small set of structures exists for all messages and files, explicit definition is more efficient, especially because the description can be omitted for local data transfer.

4.1 Services Provided to L7

- Conversion of data of type
  - integer
  - real
  - character

  (other types will be included in further versions of the CS)
- No conversion of data which has to be transparently transmitted.
- Facility to define the structure of L7-data.
- Facility to reference defined structures with identifiers.
- Facility to transmit the structure description to the destination L7-entity.

Other services such as establishment of connections, transmission of data, etc. are passed unchanged to the session layer.

4.2 Implementation

In PORTL the only structures used for communication are:

- integer, real, character
- array, record, sequential file.

Other structures such as bitree, list or cluster do not occur at the communication interface but are treated within PORTL's layers. The homogeneous structure of PORTL causes that CS is as well relieved from problems like conversion of access rights, access paths, passwords, commands or directory information.

If conversion is necessary, L6 converts data to a standard representation and then to the representation of the remote application. The introduction of a standard causes, that by having n different systems, 2n converters are needed with standard, and n(n-1) converters without it.

For describing different data representations, including the standard, Holler and Drobnik proposed in /HD75/ the use of description vectors. An integer e.g. can generally be described by the vector

$$D(I) = (n, o, n_m, c)$$

n ... total length in bits

o ... offset

n_m .. length of the number in bits

c ... { 0, if no complement is used; 1, if B-1 for negative values; 2, if B }

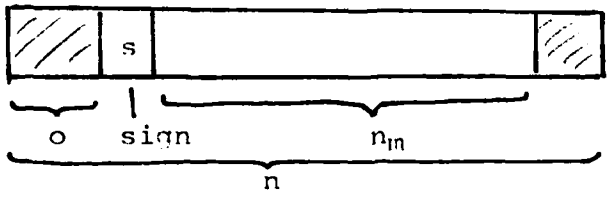


Fig. 4.1: General integer representation

According to this method e.g. a PDP-integer has the vector

$D(I)_{PDP} = (16, 0, 15, 2)$.

For CS general vectors as well as vectors of the standard representation and of all systems of the network have been specified for integer, real, character and decimal. They are sufficient to compose and describe the other structures (array, record, file).

The conversion module of layer 6 (fig. 4.2) can easily be extended for additional structures by only adding the new transformation rules and the new description vectors.

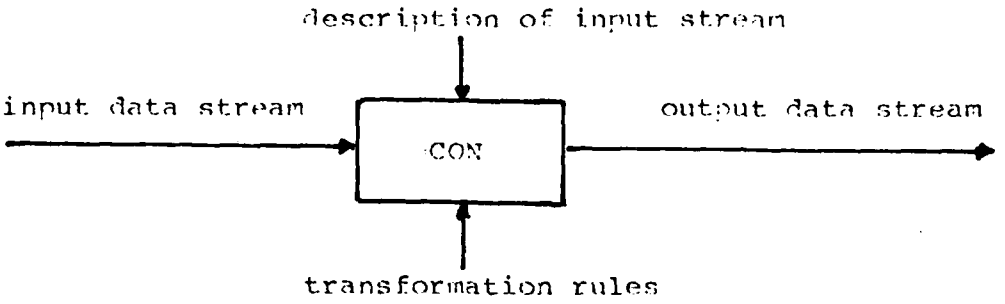


Fig. 4.2: Conversion module of layer 6

The homogeneity of the software implies that data do not have to be converted for local connections (only one programming language is used). Data possibly have to be converted, however, for connections to different computer systems. Therefore the presentation layer is implemented as a function within the transport layer.

Up to now the L6-protocol is rather simple. Only 1 type of L6-protocol-data-unit is needed, for transmitting a structure description to the remote L6-entity.

5. Session Layer

The session layer is responsible for establishing, maintaining and releasing session connections. It supports the transfer of messages and of files. It comprises flow control and error control mechanisms.

It buffers all requests coming from remote L7-entities rather than passing them to local L7-entities as "indication"-events (as mentioned in ch. 3.1). This service shall be further explained. According to Bochmann /BV78/ an (N)-service-primitive consists of 4 events (see fig. 5.1). The service requesting entity ((N+1)-entity, say) issues a "request" by calling ↓SP and gets a "confirmation". At the remote site, the (N)-entity issues ↑SP with the event "indication" and gets a "response" event.

The events "request" and "indication" presuppose the ability of handling asynchronous events. CS entities have this ability, L7-entities are not supposed to have it. Therefore only "request" and "confirmation" events exist at the L7/CS interface.

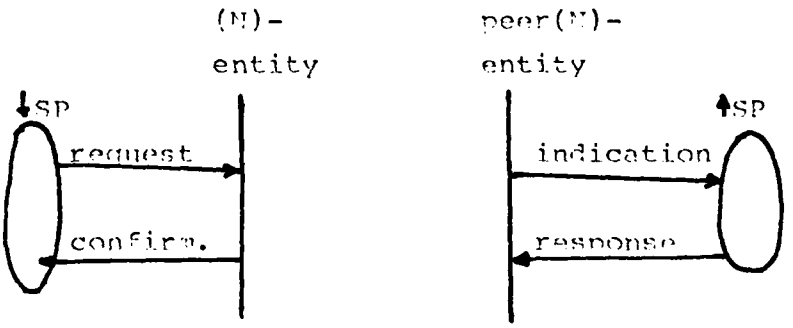


Fig. 5.1: Events of a service primitive

5.1 Session Connection and Release

A connection between 2 L7-entities is called a presentation-connection. As there is no L6-service for connection establishment/release, such requests are passed to the

session layer and presentation-connections are mapped 1:1 to session connections.

A session-connection is therefore identified just as a presentation-connection by the 2 addresses:

$(NODE_1.\ PNAME_1.\ CNO_1\ ,\ NODE_2.\ PNAME_2.\ CNO_2)$.

Within a connection request both addresses have to be specified. L5 stores all requests and establishes a connection if two L7-entities have issued requests with matching addresses. No indication and no response events are needed. This method requires an extension of the service-primitive concept. Two types of confirmation-events are needed (fig. 5.2). $CONFIRMATION_1$ indicates, that the request has been processed by the session layer (accepted or rejected). $CONFIRMATION_2$ indicates that a connection has been established. Figure 5.2 shows two cooperating session-entities ($L5_1$ and $L5_2$) handling CONNECT-service-primitives.

L5 supports generic names, i.e. NODE, PNAME, CNO and PTN may generically be specified having the meaning "any NODE", "any PNAME", etc. Another sort of generic names is PTN itself. It is a generic name for PNAME (cf. ch. 3.2).

Analogous to the establishment, a connection is released not before both L7-entities have issued release-requests (see fig. 5.3). This as well is realized by 2 types of confirmation events. The release is "soft", i.e. if one L7-entity issues a release-request, the remote L7-entity may further receive messages and files until itself makes a release-request.

CONNECT and DISCONNECT are implemented so that the applicationentity may get back control after the $CONFIRMATION_1$-event.

If a connection cannot be further maintained by the session layer, e.g. because of unrecoverable errors or because the

remote node has crashed, the existing ends of a connection change their state to "error". As there are no "indication"-events, L7-entities recognize such an error state not before completion of a service-primitive.

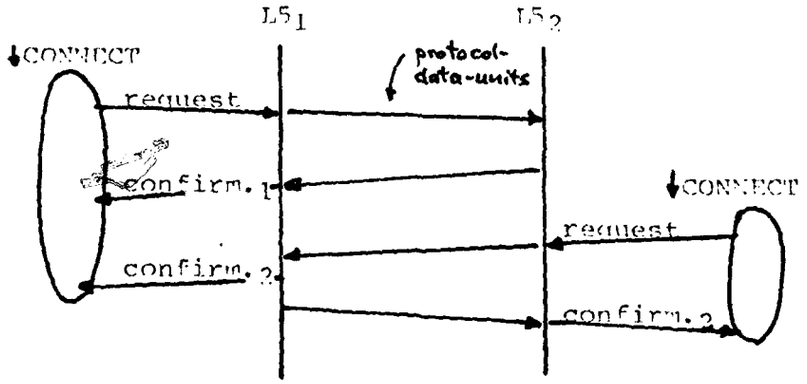


Fig. 5.2: Extended events for connection establishment

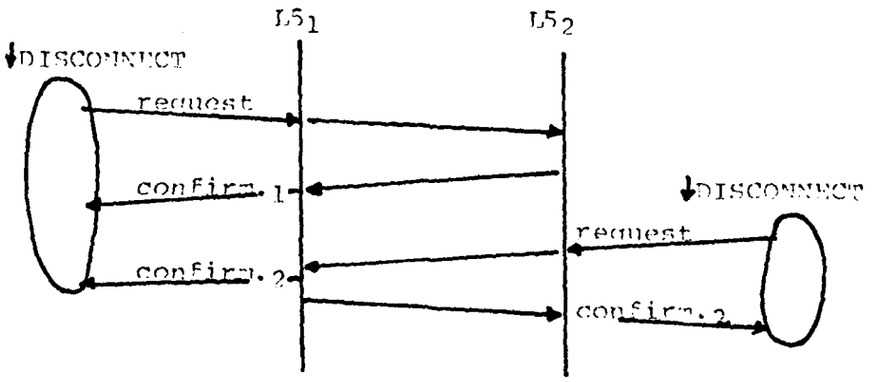


Fig. 5.3: Connection release

5.2 Data Exchange

L7-data-units are messages belonging to certain classes (which can be seen e.g. as priority classes) or files. The session layer transmits data-units maintaining their

sequence and their class. At the remote site data-units are stored until the L7-entity wants to receive one.

As L7-entities are not supposed to have flow control mechanisms, the L5-entities have to protect theirselves against data overflow.

The layer 5 flow control consists of 2 components:

- a L7/L5 interface flow control and
- a L5-intralayer flow control protocol.

If there are too many messages (files) at the remote site or in transit, the interface flow control causes that further transmit requests are rejected. In other words a L7-entity is stopped to issue further transmit requests if its peer L7-entity is too slow in receiving data-units or even has stopped to receive any.

A L5 flow control protocol is necessary because L5-entities must have the same knowledge about the number of data-units within a session-connection. If a data-unit enters or leaves a connection, the remote entity has to be informed.

A L7-entity gets no notification if its peer L7-entity has received a data-unit (see fig. 5.4). But L7-entities may inform themselves of the number of data-units which have been sent but not yet received.

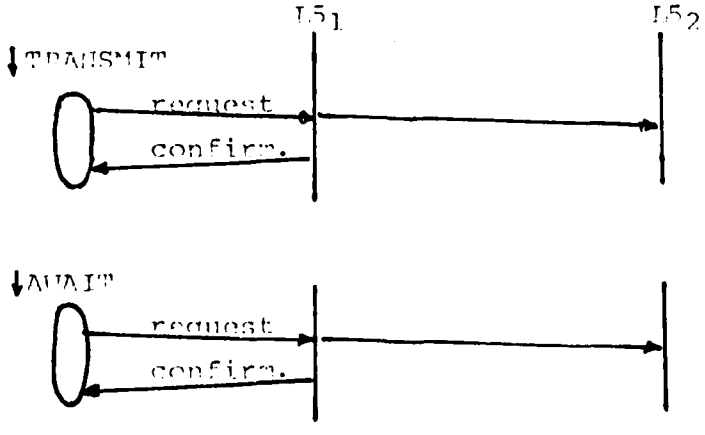


Fig. 5.4: Message exchange

5.3 The "Wait for an Event" - Service

If a L7-entity wants to receive a message from its peer L7-entity, how does it know when a message has arrived? It may loop on the "receive a message"-service-primitive (AWAIT) but this is no good programming method. The "wait for an event"-service releases L7-entities from such busy waits. It allows L7-entities to wait

- until a certain connection is established or released,
- until flow control constraints allow to transmit another data-unit,
- until a message on a certain connection or on any of the L7-entity's connections has arrived,
- until a certain file or any file has arrived.

This service is realized by a parameter of a service--primitives (the CTRL-parameter). The user thus specifies the event it wants to wait for or it specifies, that it does not want to wait.

6. Conclusion

The interprocess communication system CS was described in this paper. Requirements of the application entities (which are the users of CS), have been discussed and design criteria have been derived from them.

With its connection service (rather than datagram service), its "wait for an event" service, and its reliable data transport with data buffering at the destination, CS represents a tool by which application programmers can realize even complex communication structures in a clear, transparent and reliable way.

Up to now the described addressing mechanism of CS (see ch. 3.2) is mainly practically oriented and should not be regarded as a contribution to ISO's work. Further work has to be done in this area to clarify the requriements of the applications.

Another area which (in my opinion) needs a lot of further study is the presentation layer. Most implemented IPC systems (including file transfer systems) only use a very simple conversion mechanism. Either they only allow character strings to be transmitted or data of only one type (e.g. only integers or reals, etc.). Structured types are not considered. For our purposes the means of description vectors is appropriate to describe (simple) data structures. Whether it is suitable for other applications and for very complex structures remains to be seen.

CS is implemented on PDP11s (layers 7 to 4) and IS111s (layers 3 to 1) under the operating system RSX11M. The IS111s are used as front-end to relieve the mainframe from bit- or byte-handling communication procedures. PDP11 and IS111 are connected via direct-memory-access (DMA)- interfaces.

Acknowledgement

The author would like to thank the members of the POREL research group headed by Prof. P.J. Neuhold for all suggestions and correction hints. Special thanks to Ms. Günther for typing this paper.

References

/BV78/ Bochmann, G.V., Vogt, F.H., Message Link Protocol, Functional Specification, PIX/HLP/TAG/78/02 and in Computer Communication Review, 4/79

/BP79/ Böhme, K., Peter, G., Process Communication Structure for Distributed Systems, Proc. GI-Workshop "Kommunikation in verteilten Datenbanksystemen", Berlin, 12/79

/Boe80/ Böhme, K., The Layers 4 to 7 of an Interprocess Communication System, -Implementation Aspects-, Report No. 3/80, Institut für Informatik, University of Stuttgart, 9/80

/FN79/ Fauser, U., Neuhold, E., Transaction Processing in the Distributed DBMS-POREL, 4th Berkeley Conference, 4th Berkeley Conference, 8/79

/HD75/ Holler, E., Drobnik, O., Rechnernetze, BI Wissenschaftsverlag, Reihe Informatik 17, 1975

/PG78/ POREL, Design Specification, Reports 4/78 - 13/79 Inst. f. Informatik, Uni. of Stuttgart. (in German)

/ISO79/ Reference Model of Open Systems Interconnection, ISO/TC97/SC16 N227, 9/79

/Vo79/ Voat, F., et al., Specification of a Transport and Session Layer Protocol, Vers. 1.0, DIX/HLP/TAG/79/05, 9/79

/WF79/ Watson, R., Fletcher, J., An Architecture for Support of Network Operating System Services, 4th Berkeley Conference, 8/79

/Wa80/ Watson, R., Naming in Distributed Systems, Lecture Notes Advanced Course on Distributed Systems, Institut für Informatik, Technische Universität München, 3/80

THE TRANSPORT SERVICE OF AN INTERPROCESS COMMUNICATION SYSTEM

Klaus Böhme
Institut fuer Informatik
University of Stuttgart
Azenbergstrasse 12
D-7000 Stuttgart 1
Fed. Rep. of Germany

Abstract:

The Transport Layer of an Interprocess Communication System which is used in the DDBMS POREL is introduced. It comprises a new end-to-end error control protocol which needs a minimal number of messages in the errorfree case and avoids unnecessary retransmissions in the case of errors. This is achieved by using timers, sequence numbers, positive acknowledgement and a special resynchronization phase.

This protocol is compared with other end-to-end mechanisms: ARPA's host-to-host protocol, DoD Standard Transmission Control Protocol, and PIX Status Exchange Protocol. The use of the error control protocol is described for the tasks of the Transport Layer: connection establishment and release and data exchange.

This work has partially been supported by ERO Grant No. DAERO-79-G-0009.

1. Introduction

The Transport Service described in this paper is part of an Interprocess Communication (IPC) System, which is used in the distributed data base management system (DDBMS) POREL for data exchange between computer systems. (POREL is an experimental DDBMS implemented at the University of Stuttgart). The IPC system was designed to meet not only the requirements of a DDBMS but also for other applications which are based on communicating processes. Design criteria for POREL /see e.g. EN79, PO78/, however, essentially influenced the design of the IPC system /Boe81/.

As POREL is implemented on different computers which are connected by means of a network, we have to deal with heterogeneity. The interconnecting data network may be local or public, but we assume that it is based on packet-switching technology and that every computer is connected to it by CCITT's standard interface X.25, which is most frequently used in a heterogeneous environment. The growth of public packet-switching networks in the last 5 years justifies this assumption. There are X.25 networks in France (Transpac), U.K. (PSS), Spain (RETD), Japan (DDX), US (Telenet, Tymnet), Canada (Datapac) and FRG (DATEX-P). There are international networks as e.g. Euronet and there are connections between networks (gateways): Transpac is connected with Euronet, Telenet and Tymnet, Datapac is connected with Telenet and Tymnet /Dou80, RP80/.

It should not be concealed that X.25 has been heavily criticized. As the descriptions of the early versions of X.25 were partly inaccurate, ambiguous and incomplete, almost every network differs from the others. There are serious differences which even influence the layers using X.25, if they take advantage of features not provided everywhere, as e.g.:

- all X.25 functions in the Japanese network have end-to-end significance, normally they do not have,
- some networks provide modulo 128 packet-level numbering,
- some networks allow additional fields in CALL CONNECTED, CLEAR and CLEAR CONFIRMATION packets.

A complete list of X.25 network differences is found in /RP80/. Since 1976, when the first version of X.25 was approved, it has been further developed. The networks mentioned above are based on the versions X.25 (1976) and on X.25 (1977). The latest version of recommendation X.25 has been approved in 1980. Most of the networks will support this version by the end of 1981.

In spite of all controversies about X.25, its wide-spread use makes it convenient to take it as a basis for interconnecting computers. The differences between X.25 networks, however, makes it necessary to agree upon a common set of functions, which should be provided by every Network Layer, if X.25 should serve as a basis for the implementation of an IPC service. ISO /IS079/ or PIX /PIX79/ propose that "the Network Layer should provide for the exchange of (network-service-)data-units between transport-entities identified by network-addresses over network-connections". (The technical terms in this sentence are later explained). The Transport Layer introduced in this paper follows this recommendation. It is assumed that the network service has the following characteristics:

- Connection can be established and released between end-systems (hosts, or more exactly: transport-entities).
- The connection release phase may purge data being in transit.
- Data-units are transparently transferred and their sequence is maintained. But there may be losses and duplicates.
- Errors may or may not be notified to the Transport Layer.
- Flow control constraints may cause the Network Layer to temporarily reject data-units from the Transport Layer.

After having briefly described what is expected from the layer below the Transport Layer, we now look at the requirements of the layers above. McQuillan states in /McQ80/ that in most network projects the communication's subnetwork has been designed first and that this inside-out approach, starting from the lowest level functions and working towards the user, has not worked well. The IPC system described here was designed

top-down, beginning with the user's requirements and only realizing those functions in every layer which are needed by the layer above. For the Transport Layer, the layer above is the Session Layer which is described in /Boe80, Boe81/.

In /Boe81/ the need of session connections is discussed. We statet that for keeping messages in sequence, facilitating recovery procedures and getting transparent communication structures, in the Application Layer a connection service rather than a datagram service is better suited. Therefore the Transport Layer has to provide connections between session-entities. It further should provide for a reliable data exchange service which guarantees that data-units maintain their sequences and that no duplicated nor damaged data-units are delivered. This service should recover from lost data-units and inform the Session Layer if there are unrecoverable errors, e.g. breakdown of a remote site or of all connections to any site.

What is new?

The error control protocol of the Transport Layer was designed under the assumption that there are only few errors which cannot be recovered by X.25 and that the exchange of data-units by means of a network is considerably slower than within a computer. Therefore only a minimal number of control messages (i.e. messages which do not carry user data) should be exchanged in the errorfree (= normal) case. Some implementations use a 3-way handshake procedure (a message is acknowledged and this acknowledgement is acknowledged as well to indicate that the sender has really sent a message). In spite of being reliable, it is not very well suited because it needs 2 control-messages for every data-unit. We follow the idea of Fletcher and Watson /FW78, WF79/ to use timers rather than control messages, because timers are a local means and do not bother the network. In the errorfree case our procedure needs two timers and one acknowledging message.

In case of errors most implementations simply retransmit unacknowledged or timed out messages without considering that perhaps only the acknowledgement (ACK) has been lost. Before retransmitting a message (possibly a long one) our error control protocol assures that it was really lost, otherwise the very short ACK-message is retransmitted.

The error control protocol recovers from network generated resets, data losses, duplicates, misdeliveries and out of sequence data.

Organization of the paper

To make the reader familiar with problems concerning the Transport Layer, chapter 2 analyses some error control and flow control problems. Chapter 3 briefly describes some (in my opinion) important end-to-end protocols and discusses their advantages and disadvantages. Chapter 4 and 5 deal with our realization of the Transport Layer. The error control protocol, the connection establishment and release protocol and the data exchange protocol are introduced.

2. Transport Protocol Characteristics

A transport protocol is mainly dependent on the services it has to provide:

- Which communication structures are needed? Pairwise, multidestination, or broadcast communication; datagram- or connection-oriented?
- How can entities be addressed?
- Which error control and flow control mechanisms are needed?
- Shall multiplexing, segmenting and blocking be supported?
- Are several independent data streams and priority data-units needed?

It is further dependent on the underlying Network Layer and technology. Under the assumption that we need pairwise communication with connections based on an X.25-like Network Layer as stated in the introduction, what are the consequences for the other transport-protocol characteristics mentioned above? The goal is to design simple protocols where ideally user-data is exchanged with little control information and no additional control messages, as every control information being exchanged between entities bothers the relatively slow network and therefore decreases the capacity of the connection.

Before analysing the transport-protocol, some terms have to be defined. A service of a layer can be accessed by an abstract service-primitive (abstracting from a particularly implemented interface). According to Bochmann /BV78/ a service-primitive has 4 events associated with it (see fig. 2.1). The service requesting entity (session-entity [S1 in fig. 2.1) issues a "request" by calling tSP and gets a "confirmation". At the remote site, the transport-entity (T42) issues tSP with the event "indication" and gets a "response" event. Within a service-primitive there may be service-data. A service-data-unit is a portion of data being exchanged between adjacent layers.

All 4 events may be related by transport-protocol-data-units

(data-units which flow between transport-entities), response and confirmation, however, may only have local meaning. An example where all 4 events are related is the connection establishment mechanism used in most implementations: upon "request" a protocol-data-unit (pdu) is sent and causes an "indication" event at the remote site. The response: accept or reject is transmitted back to the requesting site within a pdu as well, causing the "confirmation" event. Two pdus are needed if all 4 events are mutually related.

The relation between service-primitive and protocol-data-unit is illustrated in fig. 2.2.

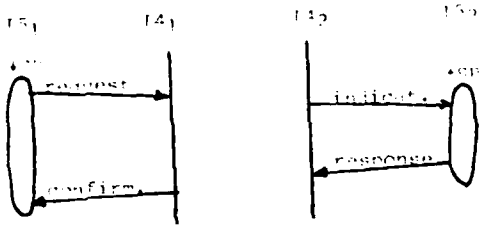
Schindler /Sch80/ has (re-)introduced the notion of service-primitve in a slightly different way. He only uses "request" and "indication" events. In his approach the events "response" and "confirmation" are the events "request" and "indication" of another service-primitive.

2.1 Flow Control and Multiplexing

Refer to fig. 2.3 for an example. Suppose that there are 2 computer systems and the session-entity $L5_1$ is connected with $L5_3$ and $L5_2$ with $L5_4$. Each connection may have its own network-connection or, as illustrated in fig. 2.3, both may be multiplexed into 1 network-connection.

Multiplexing may be necessary if only a small number of network-connections are available, possibly only 1 to each site (as in our implementation). It may be convenient if the underlying network is public, i.e. supplied by the PTT, and if its tariff structure implies that it is more cost-effective to maintain 1 network-connection for several transport-connections.

Multiplexing increases the protocol overhead. For if a session-entity wants to transmit a data-unit over a transport-connection to its peer session-entity, the local transport-entity determines the respective network connection and forwards the data-unit. The remote transport-entity receives it on that



+SP, +SP ... service-primitive initiated by $I5_1$, or by $I4_2$ resp.

$I5_1, I5_2$ ... entities of the Session layer

$I4_1, I4_2$ ... entities of the Transport layer

Fig. 2.1: The events of a service-primitive

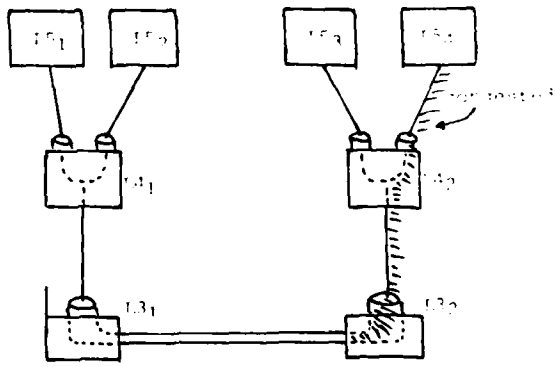
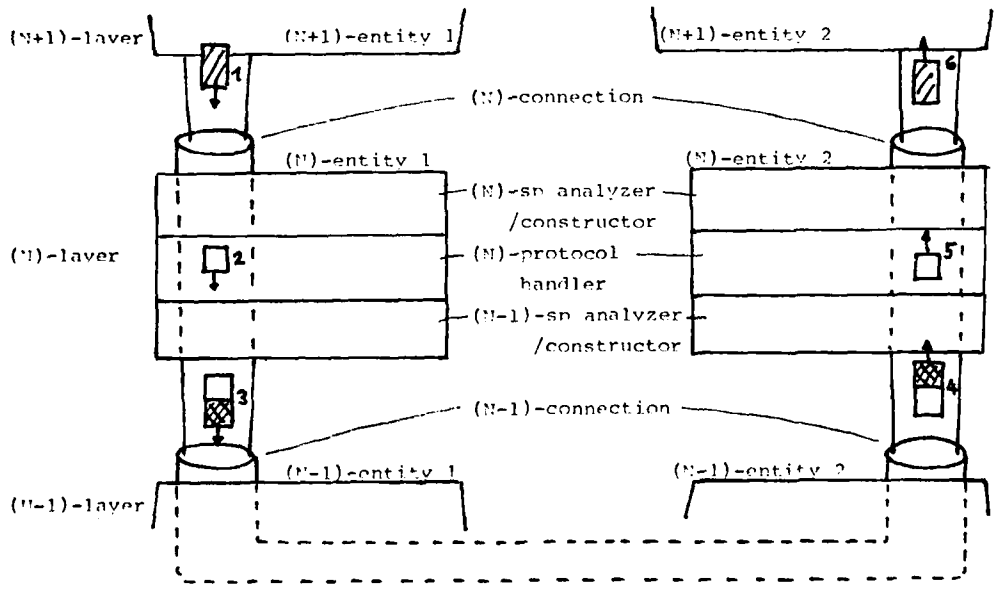


Fig. 2.3: Flow Control and Multiplexing



SP ... service primitive

▨ ... (N)-service-primitive-request

▭ ... (N)-protocol-data-unit

▩ ... (N-1)-service-primitive-request

4 ... handwritten numbers indicate points of time

Fig. 2.2: Mapping of service-primitives into protocol-data-units and their flow

network-connection, but now has to determine the resp. transport-connection and thus the correct session-entity. Therefore every data-unit has to be accompanied by a transport-connection-identifier. In the case of 1:1 mapping of transport-connections into network-connections, the network-connection determines the transport-connection and thus the resp. session-entity. No further information has to flow between transport-entities in this case.

We now will see how multiplexing influences flow-control (FC) mechanisms. FC is required to protect a message consuming entity from overflow if it is slower than the message producing entity. There may be FC between adjacent layers (interlayer FC) or between co-operating entities within <u>one</u> layer (intralayer FC). The former does not influence the transport-protocol because it is only a local means of the interface. The latter needs control information to be exchanged across the network (to indicate that the receiver is able to accept another message or to stop the sender) and therefore increases the protocol overhead.

As X.25 provides FC, is it necessary to further have FC mechanisms in the layers above? Assume that there is no multiplexing within the Transport layer. If then e.g. the $L4_2$-entity of fig. 2.3 stops receiving data-units from the L4/L3 interface (L4 stands for Transport Layer, L3 for Network Layer) and if there is an L4/L3 interlayer FC mechanism, $L3_2$'s buffers get full. This condition may be propagated to the other end of the network-connection (because of the FC of X.25) and causes the $L3_1$-entity to stop receiving data-units from the $L4_1$-entity. The same argumentation holds for the higher layers so that finally the entity of the highest layer (an application-entity) is stopped to send further data-units. Therefore the X.25 FC mechanism together with interlayer FC mechanisms are sufficient to avoid overflowing a receiving entity.

In the case of multiplexing both transport-connections into 1 network-connection as shown in fig. 2.3, $L5_4$'s rejection of

further data-units causes that $L4_2$'s buffers get full and those of $L3_2$ as well. As $L3_2$ stops accepting further data-units from the network, the connection $L5_1/L5_3$ is blocked as well, even if both entities work correctly. Therefore an independent FC within the Transport or Session layer is reasonable in this case.

2.2 Error Control

As mentioned in the introduction X.25 guarantees transmission with a rate of undetected errors close to 0. It guarantees the integrity of network-pdus (protocol-data-units) and preserves their sequence. Normally this is sufficient for the Transport Layer. But in most implementations only some parts of the X.25 protocol have end-to-end significance (i.e.occur at both ends of the connection), other parts as e.g. resets or flow control only concern the local interface and espec. X.25 does not guarantee the delivery of network-service-data-units preceding a termination- or reset-request. Therefore data-units may be lost and have to be retransmitted. Retransmission together with resets may lead to duplicates and out of sequence data-units. This can cause cooperating transport-entities to disagree upon the state of their connection and may therefore lead to deadlock situations within the Transport Layer.

Therefore the Transport Layer error control mechanism must have end-to-end significance, it has to resynchronize transport-entities and it has to recover from lost, duplicated, mis-delivered or damaged data-units. Methods for these requirements are: sequencing, acknowledgement/retransmission, timer and checksum.

In the following some problems with the data-exchange at the L4/L3 interface shall further be investigated and it shall be determined which of the error control mechanisms are suitable and necessary in the transport-protocol. It is assumed that transport-entities use sequence numbers and acknowledge pdus to detect lost or duplicated ones (fig. 2.4).

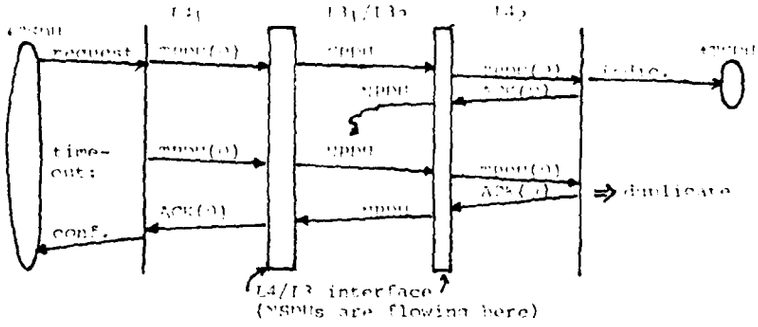


Fig. 2.4: Error control with sequence numbers

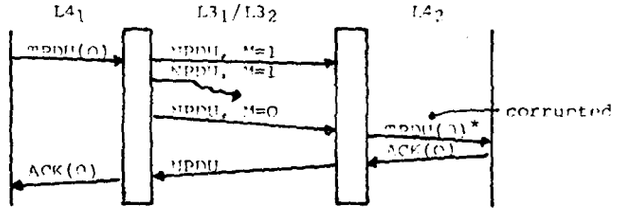


Fig. 2.5: Error situation with fragmentation of transport-pdus

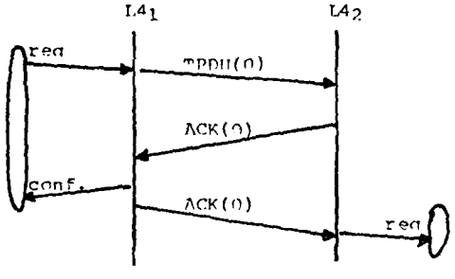


Fig. 2.6: 3-Way-Handshake-Mechanism

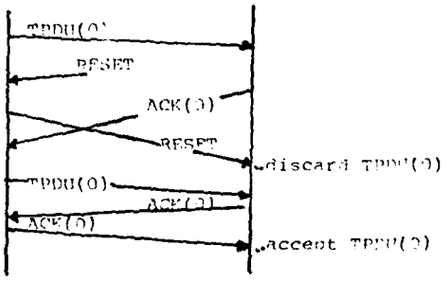


Fig. 2.7: 3-way-handshake and resets

**TSDU	...	transport-service-data-unit primitive
TPDU(0)	...	transport-pdu with sequence number 0
ACK(0)	...	acknowledgement-pdu with sequence number 0
NPDU	...	network-pdu (=data-packet)
M=1, M=0	...	more-data-bit set, reset (in a data-packet)
NSDU	...	network-service-data-unit
"RESET"	...	network generated reset

a) There is no fragmention within the Network Layer.

That is, every transport-pdu is mapped into 1 network-pdu. (It is always mapped into 1 network-service-data-unit). As the Network Layer guarantees the integrity of a network-pdu only complete transport-pdus may be duplicated or lost. Such errors occur together with L3-RESET operations.

a1) If a L3-RESET operation occurs and is notified to at least one end of the network connection (by a RESET-INDICATION service-primitive event), the L4-entity only has to initiate a end-to-end significant action to inform its peer L4-entity of the reset condition. This action may be e.g. sending a L4-RESET-pdu or clearing the network connection. No timers within the Transport Layer are needed in this case.

a2) If a L3-RESET is not necessarily noticed by the L4-entities, losses of network-service-data-units can only be recovered by timers of the transport-protocol.

b) There is fragmentation within the Network Layer.

Fragmentation is necessary if the size of a service-data-unit exceeds the maximum size of a protocol-data-unit. This is the case between Transport and Network Layer. Transport-pdus may be of arbitrarily length, whereas network-pdus mostly have a limited size which is oriented at the optimal pdu size of the network. A pdu size is optimal when the greatest effective channel capacity is obtained. Too small pdus cause too much overhead, too long pdus have to be retransmitted too often (a certain bit error rate assumed). The real capacity is decreased by contol information (protocol headers, ACK-pdus, etc.) and by retransmissions. In the case of X.25 a transport-pdu (= 1 network-service-data-unit) may be mapped into a sequence of data-packets which all except the last have the more-data-bit set.

b1) A L3-RESET operation is notified to at least 1 L4--entity. Fig. 2.5 shows that if only $L4_1$ gets a RESET-INDICATION, how can $L4_2$ know that TPDU(0)* is corrupted? Either a checksum is needed which protects the whole transport-pdu and is generated and checked by L4-entities, or a 3-way-handshake mechanism is needed. In such a mechanism a L4-entity sends a pdu and when getting an acknowledgement, acknowledges this one (fig. 2.6 and 2.7).

b2) If there is no notification of RESETs to the Transport Layer, only a checksum method is able to recover from losses of parts of a transport-pdu.

Case b) can be avoided if the L4-protocol does the fragmentation so that network-service-data-units can be transmitted within 1 network-pdu.

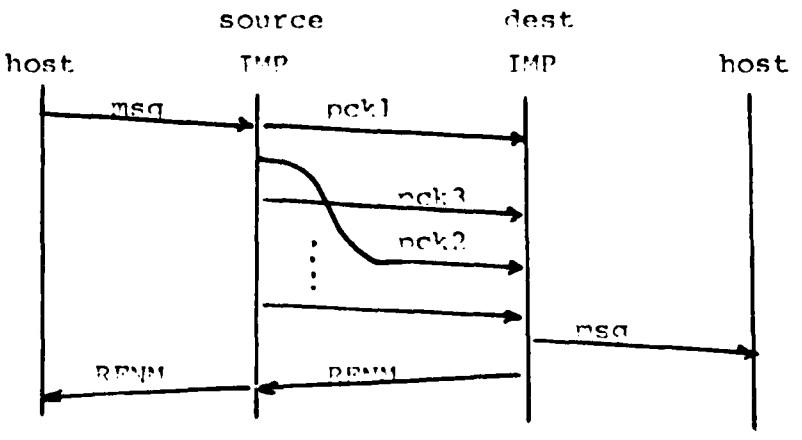
3. Realizations of End-to-End Error Control

In this chapter some end-to-end error control mechanisms are analysed. It is shown which problems they cover and which they do not.

3.1 ARPA Network Source-to-Destination Transmission Procedure

The ARPA network is one of the first highly developed packet switching networks with sophisticated error control, flow control and routing algorithms.

In the ARPA network /Mc077/ hosts are connected via IMPs (interface message processors) which are the nodes of the network. Messages (less than about 8.000 bits long) are transmitted between hosts, packets (maximally about 1.000 bits long) between IMPs. The source IMP breaks a message into several packets and sends them to the desination IMP which reassembles the packets into the message and delivers it to the destination host (see fig. 3.1). Fragmentation is used to decrease the delay of messages flowing through the network. Packets of a message are independently forwarded over arbitrary routes of the network. (Optimal flow, maximum delay, etc. are discussed in /Mc077/ or /KS75/).



msg ..... message
pcki ..... packet with sequence no. i
RFNM ..... ready for next message

Fig. 3.1 Message Transmission in the ARPA Network

Several messages may be in transit between any 2 hosts. Because of the routing algorithm packets may arrive out of order. The transmission procedure provides the following:

problem	solution
- reorder messages before delivery	- message numbering
- detect damaged or incomplete messages	- checksum over the message
- break (long) messages into smaller parts before transmission	- source IMP breaks a message into packets, destination IMP reassembles it
- reorder packets at the destination IMP	- packet numbering
- detect lost or duplicated packets	- packet numbering and timeout
- end-to-end flow control	- send RFNM back to source
- end-to-end error control	- send RFNM back to source and timeout

The end-to-end acknowledgement RFNM (ready for next message) first indicates to the source IMP that the destination IMP correctly has received a message (end-to-end error control) and second indicates that reassembly storage is allocated to receive a new message (end-to-end flow control). If a RFNM has not been received for too long a period, the source IMP times out a message number and sends a control message to the destination IMP, which has to state in a RFNM whether the message in question has arrived or not.

The ARPA source-to-destination transmission procedure does not protect the way of a message from host to host but from source IMP to destination IMP. Therefore there may be error situations which causes some problems to the higher layer protocols. E.g. what happens if a misaddressed or misrouted message is received by an IMP and is delivered to its host? Is it guaranteed that after an IMP breakdown a duplicated message is not (once again) delivered to the host? What indicates a RFNM which is

received by an TMP without being expected? Some of those problems which normally occur very infrequent are solved in other transport protocols.

3.2 DoD Standard Transmission Control Protocol

The private packet switching network ARPANET of the U.S. Department of Defense (DoD) has further been developed since its beginning in 1968. Its packet switching technology for leased lines has been applied to satellite communication, mobile radio and coaxial cable. To interconnect networks based on these technologies, the DoD has standardized an Internet Protocol and a Transmission Control Protocol (TCP) for all packet communication systems /Ce80, Po80/.

The TCP provides a reliable end-to-end communication service. It transmits segments (= some number of octetts) on established connections. It uses sequence numbers, positive acknowledgement, timeouts and checksums of segments to detect and recover from lost, duplicated, out of sequence and damaged segments.

Great care was taken that the error control mechanism is highly reliable even if crashes cause that segments or the memory of sequence numbers in use were lost or cause that duplicated or delayed segments arrive.

For connection establishment (OPEN) a 3-way-handshake procedure is used to assure that no old delayed request causes the establishment of a connection. It works as follows: a transport-entity (TE) gets an OPEN, acknowledges it, upon which the sender TE has to confirm that it really sent an OPEN-request).

As there may be rapid OPEN/CLOSE sequences, the initial sequence number for the first segment of a connection is choosen by the sender TE so that the receiver can identify delayed segments from previous incarnations of the connection. A generator is used for that which generates unique numbers

(within a period of approximately 5 hours). It is assumed that segments will stay in the network no more than 1 or 2 minutes (the maximum segment lifetime (MSL) is choosen to be 2 minutes). Therefore if sequence number memory was lost, the sender only keeps quiet for a MSL before continuing to assign sequence numbers to segments and thus no old segment will be in the network having a sequence number equal to a newly created one.

As mentioned in the introduction, 3-way-handshake is reliable but causes some overhead which, however, may be justified for the requirements of ARPANET's applications (mainly military applications). Segment exchange, however, is not based on 3-way-handshake and therefore some problems still remain. E.g. it may be possible that a TE receives a segment, delivers it to the layer above and sends back an ACK. If the ACK is lost the sender TE retransmits the segment (because it has been timed out) but if moreover the sequence number memory of the receiver gets lost, how can it detect that the incoming segment is a duplicate? Possibly the segment is delivered twice.

3.3 PIX Status Exchange Protocol

The Status Exchange (SEX) sublayer is part of PIX's Transport Layer and has mainly been developed by G.V. Bochmann /BV78, PIX79/. The SEX-protocol guarantees reliable end-to-end data exchange even with network generated resets and purges of data-units.

Contrary to most implementations each SEX-data-unit carries only one sequence number (SNO). An acknowledgement number is not used. For 2 cooperating SEX-entities it is assumed that both know the current SNO and can distinguish between "old", "current" and "new" numbers (realized e.g. by modulo 3 arithmetic). The SEX-sublayer provides only 1 service-primitive to its layer above:

S(s) .... status-exchange service-primitive with parameter s: status-data

It uses 2 protocol-data-units:

DO(N,s) ..... for status requests, N: sequence no., s: status-data

DONE(N,s) ... for status response, N: sequence no., s: status-data

The SEX-protocol detects and recovers from duplicated, lost and out of order data-units by using sequence-numbers, timers, positive ACK and retransmissions.

The mechanism is defined by table 3.1 which shows the transitions of a SEX-entity. In the errorfree case it works as

enabling conditions			actions			
active place			on local variables	interface events lower	upper	new active place (if changed)
1	1	$S_{req}(s)$	OVs:=Vs; Vs:= s; ON:= N; N:=next(N);	↓do(N,Vs)		2
2	1	↑do(n,s) and n="new"			$S_{ind}(s)$	3
3	1	↑do(n,s) and n="current"		↓done(N,Vs)		
4	1	↑done(n,s)	{ignore}			
5	2	time-out and/or reset		↓do(N,Vs)		
6	2	↑done(n,s)and n="current"			$S_{conf}(s)$	1
7	2	↑done(n,s)and n="old"	{ignore}			
8	2	↑do(n,s)and n="current"		↓ done(N,Vs)	$S_{conf}(s)$	1
9	2	↑do(n,s)and n="old"		↓ done(ON,OVs)		
10	2	↑do(n,s)and n="new"	{ignore}			
11	3	$S_{resp}(s)$	OVs:=Vs; Vs:= s; ON:=N N:=next(N)	done(N,Vs)		1

Vs: "current" value of the status data
OVs: "old" value of the status data
N: "current" sequence number
ON: "old" sequence number
next(N) = (N+1)modulo 3

Tab. 3.1: SEX transition table

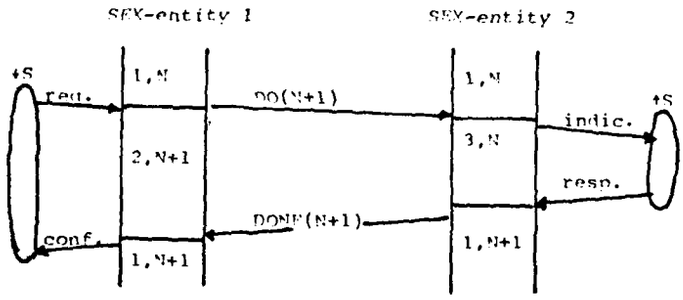
follows (see fig. 3.2):

- A Sex-entity gets a ↓S-request from above. It sends a DO-pdu with a "new" sequence number.
- The peer entity receives the DO, sends an ↑S-indication to the entity above it and waits for ↓S-response.
- Upon reception of ↓S-response, it sends DONE.
- The peer entity receives DONE and finishes the service-request with ↑S-confirmation.

Some error situations shall be discussed. If a DO- or DONE-pdu is lost, the DO is timed out and retransmitted. An example with many retransmissions is shown in fig. 3.3. Three pdus have to be retransmitted in this case. As it is assumed that DONE not only acknowledges a DO-pdu but also carries status-data, it cannot be ommited and be replaced by a subsequent DO-pdu.

3.4 Summary of chapter 3

3-way-handshake is the most reliable procedure if data-units may be lost but it produces most overhead too because every data-unit needs 2 control-messages for acknowledgement. In most protocols unacknowledged data-units are blindly retransmitted even if only the ACK was lost.



↓↑S ... SEX-service-primitive
i,N ... state of a SEX-entity: i=1 ... ready state
i=2 ... call request state
i=3 ... called request state
N ... value of the sequence number
DO(N) ... DO-pdu with sequence no. N
DONE(N).. DONE-pdu

Fig. 3.2: Status exchange in the errorfree case

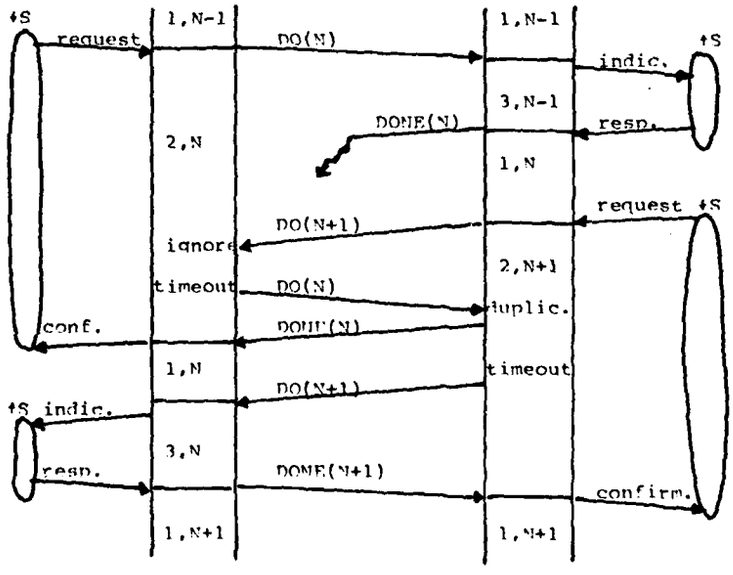


Fig. 3.3: Error situation in the SEX-protocol

4. The Transport Protocols

4.1 The Error Control Protocol

The main design criteria for error control in the Transport Layer are

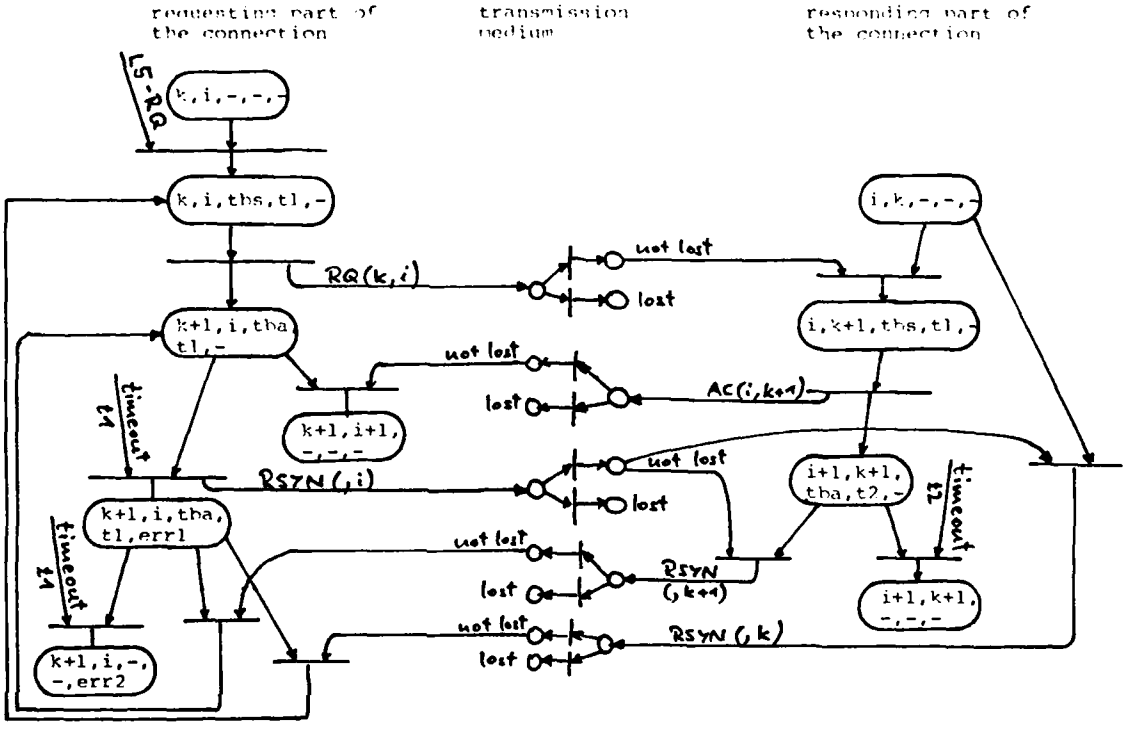
- reduce transmission overhead for the normal (errorfree) case, i.e. no extra acknowledgement pdus
- use of a window technique (several consecutive pdus are acknowledged together)
- use of timers instead of handshake procedures, to save additional ack-pdus.

In our implementation all transport-pdus are sequenced and have to be acknowledged by sending back the next expected sequence number. This number indicates that all pdus with a sequence number less than the retransmitted one have correctly been received. Transport-entities check each pdu's sequence number against the expected one. By inequality the pdu is ignored.

A timer t1 watches that an acknowledgement arrives in time. If a timeout of t1 occurs, the resynchronisation phase is started which determines which pdus have to be retransmitted. The resynchronisation phase is also initiated by a RESET of a network entity.

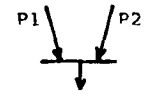
This mechanism is illustrated by a Petri-Net (see fig. 4.1). It is based on a state vector of the connection and needs two timers t1 and t2. The state vector of both ends of the connection is composed of

- the sequence number of the next pdu to be sent,
- the sequence number of the next expected pdu,
- a substate indicating whether a pdu has to be sent (tbs) or to be acknowledged (tba)
- the responsible timer (t1 or t2)
- a substate which counts consecutive errors (err1, err2, ...).



RQ(k,i) ... request-pdu with seq. no. k and ACK-no. i
AC(k,i) ... acknowl.-pdu - " -
RSYN(,j) ... resynchronisation-pdu with ACK-no. j

a,b,c,d,e — place (conditon) with state vector of (one end) of the connection
- a ... sequence no. of next pdu to be sent
- b ... sequence no. of the next expected pdu
- c ... substate indicating whether a pdu has to be sent (c=ths) or to be acknowledged (c=tha)
- d ... responsible timer (t1 or t2)
- e ... errorcount



transition (fires dependent of its conditions p1 and p2)

nondeterminism: t1 or t2 (not both) may fire dependent of the event which happens

Fig. 4.1: The error control mechanism

E.g. (k+1, i, tha, t1, err1) means that the next send number is k+1, i is the number of the next expected pdu, the pdu k has to be acknowledged, the responsible timer is t1 and 1 error has occured.

Fig. 4.1 shall further be explained. Beginning with a state where both ends of the connection have consistent sequence numbers (k, i), a request arrives at entity 1 from the layer above (L5-RQ). The pdu RQ(k, i) is sent and has to be acknowledged by an AC-pdu. Timer t1 is started. If there are transmission errors (RQ-pdu or AC-pdu is lost), t1 runs down and the resynchronisation phase is started. An unnumbered RSYN-pdu together with the next expected number is sent and has to be acknowledged by a RSYN-pdu. Again t1 watches that the acknowledging RSYN-pdu arrives in time. If not, 2 consecutive errors have occured and in this implementation the connection is supposed to be uncorrectably down. The L5-entity is notified. Otherwise the incoming RSYN-pdu indicates which sequence number is expected by the remote end, implying which pdus have to be repeated.

If an AC-pdu is sent by entity 2, the timer t2 is started. If it does not arrive at entity 1, a t1-timeout occurs and a RSYN-pdu is sent by entity 1. Otherwise the timeout t2 indicates, that the AC-pdu must have arrived at entity 1. The meaning of t2 is, that during the time t2, entity 1 may protest against the lost of its pdu or of the AC-pdu.

4.2 The Connection Protocol

Establishment

A transport-connection between 2 session-entities is established if both entities have issued matching requests, i.e. if the specified addresses correspond with each other. At the L4/L5 interface the connection is referenced by a hierarchic address (NODE, PNAME, CNO), where NODE identifies the computer site, PNAME the process name (entity name), and

CNO the connection number. Addresses may be specified generically, if the actual value is not known. (For details of the addressing mechanism see /Boe81/).

Between transport-entities a connection is referenced by a unique transport-connection-number (TCNO). Uniqueness is obtained if e.g. the set of TCNOs is totally ordered and each transport-entity has its own subset of numbers which is disjunct with the set of another L4-entity.

If a session-entity issues a request to establish a transport-connection (TCONRQ, see fig. 4.2) and if this request is (locally) accepted by the referenced L4-entity, the L4-entity transmits a L4-protocol-data-unit (L4CONRQ) to the peer transport-entity. L4CONRQ contains a TCNO which was not currently used. For transmission of the pdu, a network-connection is used which exists between the 2 related L4-entities. Such a network-connection possibly has first to be established.

The answer to a L4CONRQ-pdu is an accept- or reject-pdu (L4CONAC, L4CONRJ) which serves as an acknowledgement. If there are 2 accepted requests with matching addresses, a transport-connection is established. It is identified by the minimum of the two TCNOs which were used within the L4CONRQ-pdus. This algorithm needs no additional exchange of pdus between L4-entities to agree upon a unique TCNO (even if both L4CONRQs have overlapped).

If 2 matching requests have used different network-connections, the L4-entities have to agree upon 1 connection and can possibly release the other one. The same algorithm has to be implemented in the L4-entities so that they know which one (e.g. the least one, if all L4-entities are totally ordered) has to release a connection and which connection shall further be used.

Release

For transport-connection release both session-entities have to issue clear-requests (fig. 4.3). The peer L4-entity is informed of a clear-request by a L4CLRPO-pdu. Each L4CLRPO-pdu has to be acknowledged by a L4ACK-pdu or piggy-backed within another pdu.

The release is graceful, i.e. no data-units are purged as long as only one side has closed the connection. Therefore a session-entity may continue to receive pdus if it is in the rdrq (remote disconnect request) or rdac (remote disconnect accept) state (cf. fig. 4.3) until it issues a clear-request as well. This release mechanism is oriented at the user's requirements: if one side has terminated data exchange and does not expect further data-units, it may close the connection possibly before the other side has received all data-units. In mechanisms where data-units are purged if one side closes the connection, the user entities have to exchange messages prior to closing of a connection to be shure that the both sides have finished data exchange.

4.3 The Data Exchange Protocol

Every request to send a transport-service-data-unit causes a L4DATA-pdu to be sent over the transport-connection containing the service-data-unit. Each L4DATA-pdu is acknowledged by the peer-transport-entity (fig. 4.4).

For file transmission, the file is split into several data-units which are transmitted as L4DATA-pdus. The peer L4-entity has to know which one is the first pdu of a file transfer and which one is the last. This is attained by a L4FILFW-pdu and a L4FILEW-pdu. Both pdus have to be acknowledged (as well as all L4DATA-pdus).

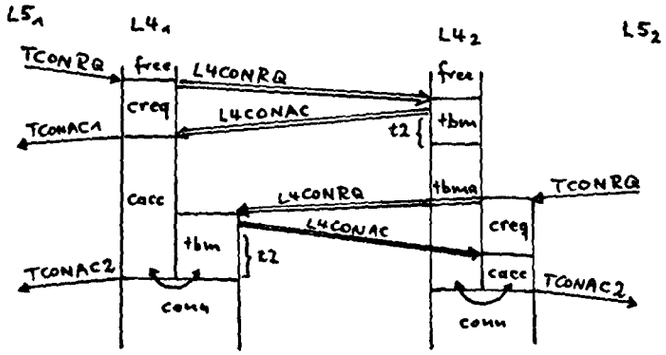


Fig. 4.2: Transport Connection Establishment

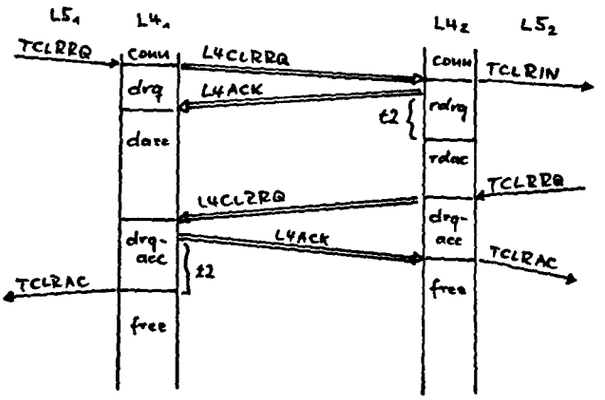


Fig 4.3: Transport Connection Release

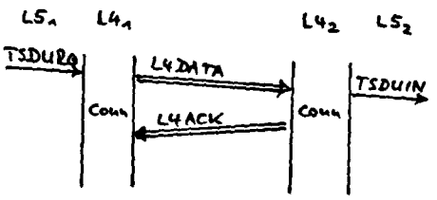


Fig. 4.4 : Message exchange

Legende:

$L5_i$	... entity i of layer 5
$L4_i$	... entity i of layer 4
→	... service-primitive
⇒	... protocol-data-unit
↻	... match of 2 connect-requests

L4-service-primitives

TCONRQ	initiate-transport-connection-request event
TCONAC1	- " - -accept event type 1
TCONAC2	- " - -accept event type 2
TCLRRQ	clear-transport-connection-request event
TCLRIN	- " - -indication event
TCLRAC	- " - -accept event
TSDURQ	transport-service-data-unit-request event
TSDUIN	- " - -indication event

L4-protocol-data-units

L4CONRQ	connect-request pdu
L4CONAC	connect-accept pdu
L4CLRRQ	clear-request pdu
L4DATA	data pdu
L4ACK	acknowledgement pdu

States of the transport-connections

free	free state
creq	connect request state
cacc	connect accept state
tbm	to be matched state
tbma	tbm state after timer t2 ran down
conn	connected state
drq	disconnect request state
rdrq	remote disconnect request state
dacc	disconnect accept state
drqacc	2nd diconnect request was issued
rdac	rdrq after t2 ran down

5. Conclusion

Transport protocols and especially their end-to-end error control mechanisms have been discussed. For an X.25-like Network Layer a new end-to-end error control mechanism has been introduced, which serves as a basis for the transport protocols: connection establishment and release and data exchange.

The main advantages of this error control are that messages are not blindly retransmitted if only an ACK was lost and that a second timer at the receiver's side gives a reliability close to 3-way-handshake.

This protocol can further be optimized. The notification of network-RESETs may be used on both sides to recover from possibly lost pdus and initiate the resynchronisation phase without waiting on the expiration of a timer. Better efficiency may also be obtained if a possibly lost short data-unit is immediately retransmitted within a RSYN-message.

The entire IPC system, where the Transport Layer is one part of it, is implemented in PASCAL and Assembler on PDP11s and LSI11s under the operating system RSX11M. The LSI11 are used as front-end and comprise the Network, Link, and Physical Layer. They are connected with the PDP by direct-memory-access interfaces.

Acknowledgement

The author would like to thank the members of the PORTL research group headed by Prof. E.J. Neuhold for many valuable suggestions which have clarified this paper. In addition special thanks to Ms. Günthör for typing this paper.

References

/BV78/ Bochmann, G.V., Vogt, F.H., Message Link Protocol, Functional Specification, PIX/HLP/TAG/78/02 and in Computer Communication Review, 4/79

/Boe80/ Böhme, K., The Layers 4 to 7 of an Interprocess Communication System, -Implementation Aspects-, Report No. 3/80, Institut für Informatik, University of Stuttgart, 9/80

/Boe81/ Böhme,K., An Interprocess Communication Service Applied in a Distributed Data Base System Techn. Report, Institut für Informatik, University of Stutgart, 3/81

/Ce80/ Cerf, V.G., Protocols for Interconnected Packet Networks, ACM, Computer Comm. Review, 10/80

/FN79/ Fauser, U., Neuhold, E., Transaction Processing in the Distributed DBMS-POREL, 4th Berkeley Conference, 4th Berkeley Conference, 8/79

/FW78/ Fletcher, J.G., Watson, R.W., Mechanisms for a Reliable Timer Based Protocol. Proceedings Computer Network Protocols, Liège, 2/78

/ISO79/ Reference Model of Open Systems Interconnection, ISO/TC97/SC16 N227,8/79

/KS75/ Kimbleton, S.P., Schneider G.M., Computer Communication Networks: Approaches, Objectives, and Performance Considerations. In ACM Computing Surveys, 9/75

/McQ77/ McQuillan, J.M., Walden, D.C., The ARPA Network Design Decisions. In Computer Networks 8/77

/McQ80/ McQuillan, J.M., Local Network Technology and the Lessons of History.
In Computer Networks, 10/80

/PIX79/ Vogt, F., et al., Specification of a Transport and Session Layer Protocol, Vers. 1.0, PIX/HLP/TAG/79/05, 9/79

/PO78/ POREL, Design Specification, Reports 4/78 - 13/78
Inst. f. Informatik, Uni. of Stuttgart.
(in German)

/Po80/ Postel, J., (ed.), DoD Standard Transmission Control Protocol.
ACM, Computer Communication Review, 10/80

/Pou80/ Pouzin, L., Recent Developments in Data Networks.
In DATACOMM Conference Proceedings, Geneva, 6/80

/RP80/ Rybczynski, A., Palframan, J., A Common X.25 Interface for Public Data Networks.
In Computer Networks, 6/80

/Sch80/ Schindler, S., Distributed Abstract Machine.
In Computer Communications, 10/80

/WF79/ Watson, R., Fletcher, J., An Architecture for Support of Network Operating System Services,
4th Berkeley Conference, 8/79

THE IPC INTERFACE TO THE APPLICATION LAYER

1. Service-Primitive Parameters

CTRL	= nowait: control is returned to the (service-requesting) L7-entity as early as possible = wait: it is waited until a certain event has happened
CNO	connection number; to identify a process' connection
RET	return code; indicates if a service primitive was successfully completed or not
STAT	state of the connection
PTN	process-type-name
RPTN	process-type-name of the remote process
PID	process-identification consists of 3 fields: NODE: identifies the computer system PNAME: process-name generated by the operating system CNO: connection number
RPID	identification of the remote process (same structure as PID)
PRIO	priority class of a message
UDATA	user-data
FID	format identifier, was declared in a DECLF-service-primitive
CNOSET	set of connection numbers
LFNAME	local filename
RFNAME	remote filename
S	number of messages/files sent but not yet received by the remote entity
R	number of messages/files sent by the remote entity but not yet received

2. Connection Establishment

| CSBEGIN, →CTRL, →CNO, ←RET, ←STAT |

Before any connection can be established with a CONNECT-primitive, CS once has to be called with a CSBEGIN primitive. After this CS knows the calling process and has reserved communication buffers. CSBEGIN and CSEND (cf. chap. 5) are the first and the last primitive which have to be called.

CTRL	not used
CNO	not used
RET	1 or -4
STAT	not used

| CONNECT, →CTRL, →CNO, ←RET, ←STAT, →PTN, ←→RPTN, ←→RPID |

A connection with the number CNO is to be established to the process RPID of type RPTN. The connection is existing, if RPID has issued a corresponding (matching) CONNECT. Values which have been generically specified are returned by the CS after connection establishment.

CTRL	= nowait: wait until CS has accepted or rejected the CONNECT-request = wait: wait until the request is rejected or the connection is existing
CNO	
RET	cf. 1.
STAT	
PTN	own process-type-name
RPTN	remote process-type-name generic value: RPTN = ' ' meaning "a process of any type"
RPID	remote process-identification generic values: NODE = 0 : any computer system PNAME = 0 : any process CNO = 0 : any connection of the process

3. Data Exchange

3.1 Format Declaration

DECLF,→ CTRL, →CNO,← RET, ←STAT,→FID, →FORMAT

The structure of a data-unit (message, file) is made known to the CS and can further be referenced by its identification FID. It is described by a character string, whose syntax is FORTRAN-like (e.g.: '1C5,10I' means 5 character, 10 integer).

CTRL — not used

CNO
- = 0: global declaration, valid for all connections of the process
- = 0: the declaration is valid only for the specified connection

RET
STAT — cf. 1.

FID — identifies the declared structure (unique within the scope of the process or the scope of the specified connection). Note: FID=0 and FID=-1 have special meanings.

FORMAT — consists of 2 fields:

LNG: length of the format-string in bytes

FSTRING: character string which describes the structure of a data-unit. It may contain:

- nI for integer
- nF for real
- nCi for character (n,i integer)

| DECLFC, →CTRL, →CNO, ←RET, ←STAT, →SETOFFID |

The DECLFC-primitive is necessary if the structure of the messages has been defined with a PASCAL-case-record. Every message correspondends to only 1 case of the record definition. The structure of each case has to be declared within a DECLF-primitive. Therefore a FID exists for each case. All FIDs, i.e. all cases have to be combined with a DECLF-primitive.

CTRL	not used
CNO	see DECLF-primitive
RET	cf. 1.
STAT	
SETOFFID	the FIDs of all FORMATs which shall be combined, they have to be previously defined with DECLFs

3.2 Message Transfer

| TRANSMIT, →CTRL, →CNO, ←RET, ←STAT, →PRIO, →FID, →UDATA |

The data-unit UDATA of the structure FID and with priority PRIO is to be transmitted over the connection CNO.

CTRL	= nowait: no waiting if the data-unit cannot be transmitted by CS (probably because of flow control constraints) = wait: wait until the data-unit can be transmitted.
CNO	
RET	cf. 1.
STAT	
PRIO	priority class of UDATA
UDATA	user-data-unit
FID	format-identifier 0: FID corresponds to a FORMAT which was declared in a DECLF primitive = 0: no FORMAT exists, UDATA is transmittted without conversion (tranparent mode)

| AWAIT, →CTRL, →CNO, ←RET, ←STAT, →PRIO, ←UDATA |

A data-unit with priority PRIO is to be received on the connection CNO and is to be copied into the variable UDATA.

CTRL	= nowait: no waiting if there is no data-unit of priority PRIO = wait: wait until there is a data-unit
CNO	
RET	cf. 1.
STAT	
PRIO	priority class of UDATA
UDATA	variable in which the received data-unit is to be copied.

| AWAITL, →CTRL, →CNO, ←RET, ←STAT, →PRIO, →CNOSET, ←UDATA |

A data-unit with priotrity PRIO is to be received on any of the connections specified in CNOSET.

CTRL	= nowait: no waiting if there is no data-unit = wait: wait until there is a data-unit on any of the connections specified in CNOSET
CNO	number of the connection where the data-unit was received (return-parameter!)
CNOSET	set of connection numbers
RET	
STAT	cf. AWAIT service-primitive
PRIO	
UDATA	

3.3 Filetransfer

TRANSMITF, →CTRL, →CNO, ←RET, ←STAT, →LFNAME, →RFNAME, →FID

The file LFNAME is to be transmitted over the connection CNO. It gets the name RFNAME at the destination.

CTRL = nowait: no waiting if the file cannot be transmitted by CS
= wait: wait until the file can be transmitted

CNO

RET cf. 1.

STAT

LFNAME local filename

RFNAME remote filename, may be generic: ' '. CS then generates a filename at the remote site.

FID format identifier
0: a FORMAT has been declared in a DECLF
= 0: transparent mode
-1: implicit description mode, i.e. the file consists of records which first contain a description field a then the value

AWAITF, →CTRL, →CNO, ←RET, ←STAT, ↔FNAME, ←FORMAT

The file FNAME is to be received on the connection CNO.

CTRL = nowait: no waiting if there is no file
= wait: wait until there is a file having the name FNAME

CNO

RET cf. 1.

STAT

FNAME name of the file which is to be received
generic: FNAME = ' ' : any file is to be received

FORMAT — description of the contents fo the transmitted file as declared in a DECLF-service-primitive by the remote process (cf. 3.)
LNG=0 : transparent mode, no FSTRING exists
LNG=-1: implicit description mode

4. Information of a Connection

INFORM, →CTRL, ↔ CNO, ←RET, ←STAT, ↔ RPTN, ↔ RPID, ←S, ←R

Parameters of the connection CNO or of the connection to RPID resp. are returned.

CTRL — not used

CNO — connection number
if not specified (CNO = 0), RPID has to be specified and CNO is returned

RET
STAT
RPTN
RPID

cf. 1.

S — number of messages/files which were sent but not yet received by the remote process

R — number of messages/files which were sent by the remote process but not yet received

5. Connection Release

DISCONNECT, →CTRL, →CNO, ←RET, ←STAT, ←S, ←R

A connection is to be released. Locally the state of the connection changes to "dreq" at the remote site to "rdreq". As soon as the remote process has also issued DISCONNECT, the connection is non-existant.

CTRL = nowait: no waiting
= wait: wait until the connection is released

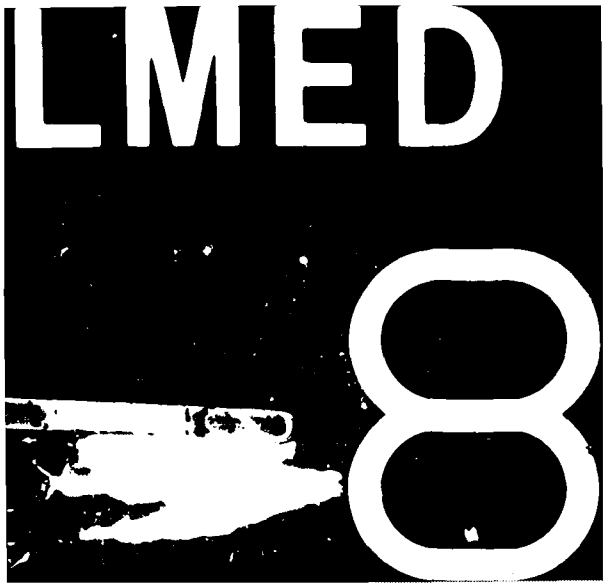
CNO
RET — cf. 1.
STAT

S — cf. 4.
R

CSEND, →CTRL, →CNO, ←RET, ←STAT

To finish the interaction with CS, the user process has to call the CSEND-primitive.

CTRL
CNO — not used
RET
STAT


LMED
8